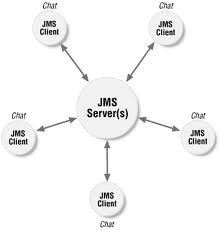
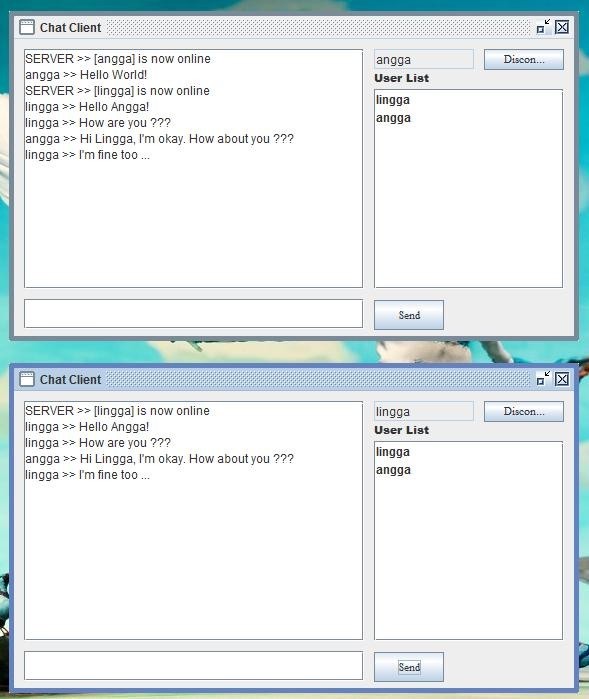
**Multi User Chat System Using Java**



* Multi User Chat System is an application through which the user can communicate with other users connected in the same network area (LAN). This works under any operating system and is programmed in java.
* To establish a communication between the systems, we need simple socket connections in order to connect them in a network.
* Socket programming uses the client socket and server socket methods to connect the local host to the named host and port
* The communication between various users is done using server client model.
* Several client machines are connected to their dedicated server ports and communication is established.
* We use the lowest level of networking techniques available in java, though there are many ways of connecting client with the server.
* Hence, we need simple classes of networking package in order to establish a server client prototype.

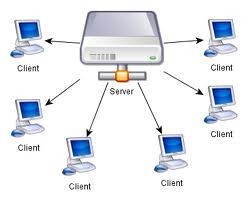
What is a server??

* A **server** is a system ([software](http://en.wikipedia.org/wiki/Computer_software) and suitable [computer hardware](http://en.wikipedia.org/wiki/Computer_hardware)) that responds to requests across a [computer network](http://en.wikipedia.org/wiki/Computer_network) to provide, or help to provide, a [network service](http://en.wikipedia.org/wiki/Network_service).
* Servers can be run on a dedicated computer, but many networked computers are capable of hosting servers. In many cases, a computer can provide several services and have several servers running.
* Servers are computer programs running to serve the requests of other programs, the clients. Thus, the server performs some task on behalf of clients. The clients typically connect to the server through the network but may run on the same computer. In the context of Internet Protocol (IP) networking, a server is a program that operates as a socket listener.

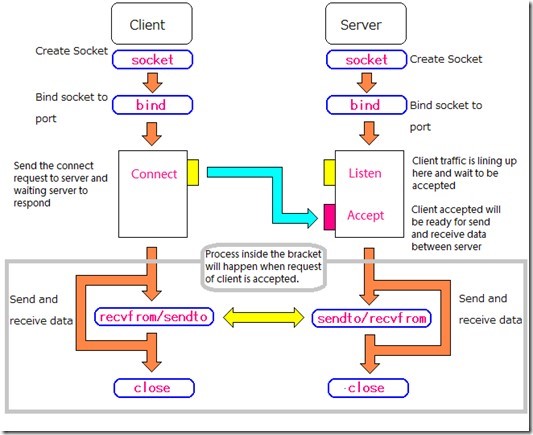


Client:

Client is a user system which uses its server to get its program or action executed.



* To have a basic server client connection through java programming language, we need the concepts of java as such as,
* 1. Network packages
* 2. Thread classes
* 3. AWT tool kit
* 4. Event handling.



Step by step modules…

* Listener class
* While-Accept loop
* Per-Thread class
* While-Read/Write loop (Server side)
* Removing dead connections
* Client class
* While-Read/Write loop (Client side)

1.What does the server do?

- Stand alone program (JVM)

2.Listening to a port

After a server is ready, get ready to receive incoming connections.

3. Sockets

- Communications pass through sockets. Socket has an InputStream and an OutputStream.

• PORT

// Constructor and while-accept loop all in one. public Server( int port ) throws IOException {

// All we have to do is listen

listen( port ); }

SERVER

// start listening on the port

ServerSocket ss = new ServerSocket( port );

// loop forever while (true) { // get a connection

Socket newSocket = ss.accept();

// deal with the connection

The main() routine

// Main routine

// Usage: java Server >port< static public void main( String args[] ) throws Exception {

// Get the port # from the command line int port = Integer.parseInt( args[0] );

// Create a Server object, which will automatically begin // accepting connections.

new Server( port );

4.Serialization of incoming requests - multithreading.

private void listen( int port ) throws IOException {

// Create the ServerSocket ss = new ServerSocket( port ); // Tell the world we're ready to go

System.out.println( "Listening on "+ss ); // Keep accepting connections forever while (true) {

// Grab the next incoming connection

Socket s = ss.accept();

// Tell the world we've got it

System.out.println( "Connection from "+s );

// Create a DataOutputStream for writing data to the

// other side

DataOutputStream dout = new DataOutputStream( s.getOutputStream() );

// Save this stream so we don't need to make it again outputStreams.put( s, dout );

// Create a new thread for this connection, and then forget

// about it

new ServerThread( this, s );

}

}

5. The communications protocol

* When a user types something into their chat window, their message will be sent as a

string through a DataOutputStream.

* When the server receives a message, through a DataInputStream, it will send this same

message to all users, again as a string through a DataOutputStream.

* The users will use a DataInputStream to receive the message.

6. Removing dead connections

// The connection is closed for one reason or another,

// so have the server dealing with it server.removeConnection( socket );

Now, to the client side 1. Use an interface.

2. Connect to the server // Connect to the server try {

// Initiate the connection socket = new Socket( host, port ); // We got a connection! Tell the world

System.out.println( "connected to "+socket );

// Let's grab the streams and create DataInput/Output streams

// from them din = new DataInputStream( socket.getInputStream() ); dout = new DataOutputStream( socket.getOutputStream() ); // Start a background thread for receiving messages new Thread( this ).start();

} catch( IOException ie ) { System.out.println( ie ); } }

The While-Read/Write loop (Client side)

// Background thread runs this: show messages from other window

public void run() {

try {

// Receive messages one-by-one, forever

while (true) {

// Get the next message

String message = din.readUTF(); // Print it to our text window

ta.append( message+"\n" );

}

} catch( IOException ie ) { System.out.println( ie ); } }

Pretty simple. Each incoming message gets displayed in the text display window, and then the loop goes back to waiting for the next message.

Limitations:

Cant add a new client when a communication is under process.

***Queries please***

